Algorithms & Complexity
Lecture 4: Recursive and Greedy Algorithms

October 8, 2019

CentraleSupélec / ESSEC Business School

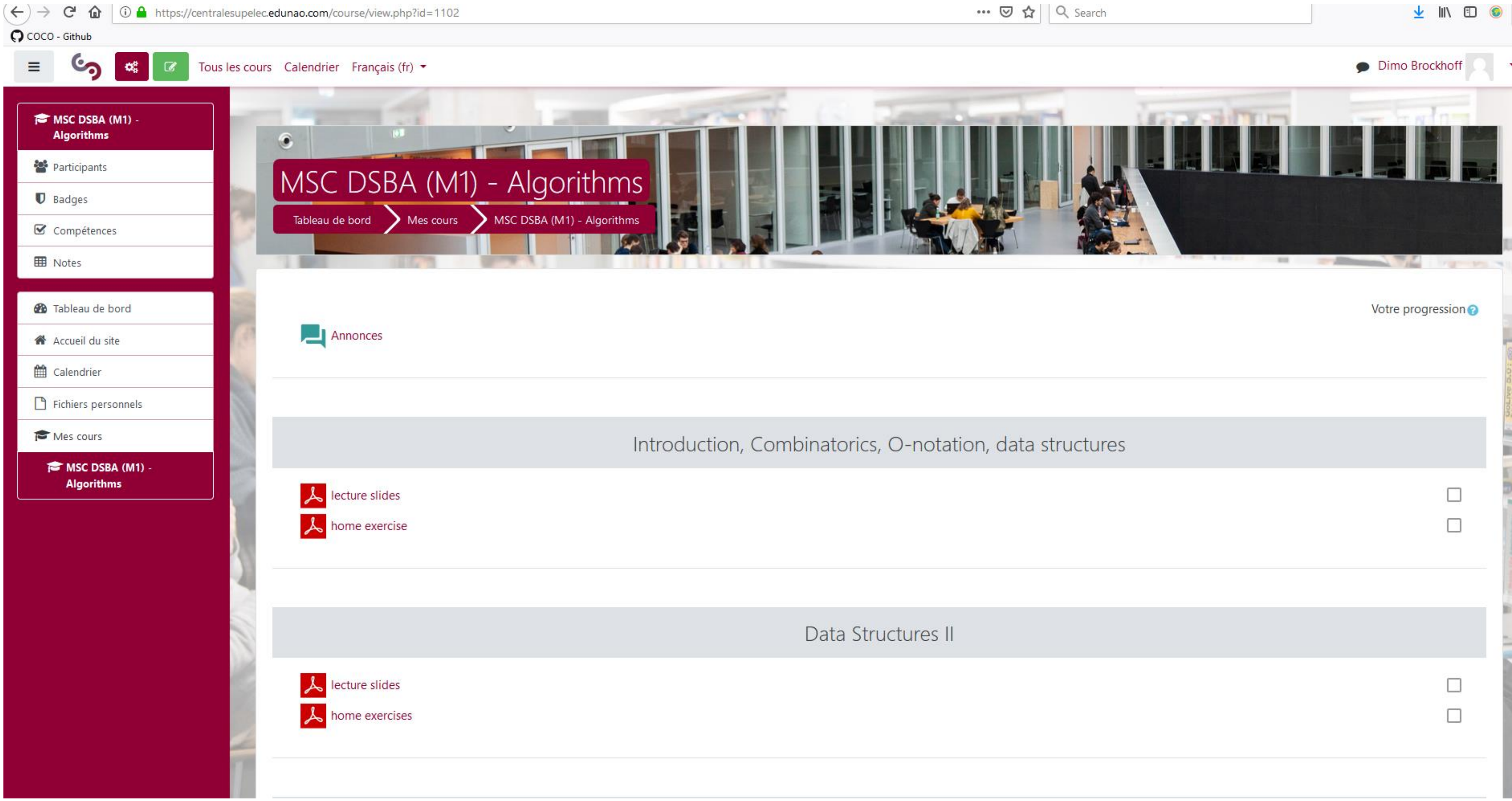
Dimo Brockhoff

Inria Saclay – Ile-de-France

Course Overview

Thu		Topic
Thu, 12.09.2019	PM	Introduction, Combinatorics, O-notation, data structures
Tue, 24.09.2019	PM	Sorting algorithms I
Tue, 1.10.2019	PM	Sorting algorithms II, recursive algorithms
Tue, 8.10.2019	PM	Recursive and Greedy Algorithms
Tue, 15.10.2019	PM	Dynamic programming
Thu, 31.10.2019	AM	Randomized Algorithms and Blackbox Optimization
Tue, 5.11.2019	PM	Complexity theory I
Tue, 26.11.2019	PM	Complexity theory II
Tue, 17.12.2019	AM	Exam (written)

Announcement 1

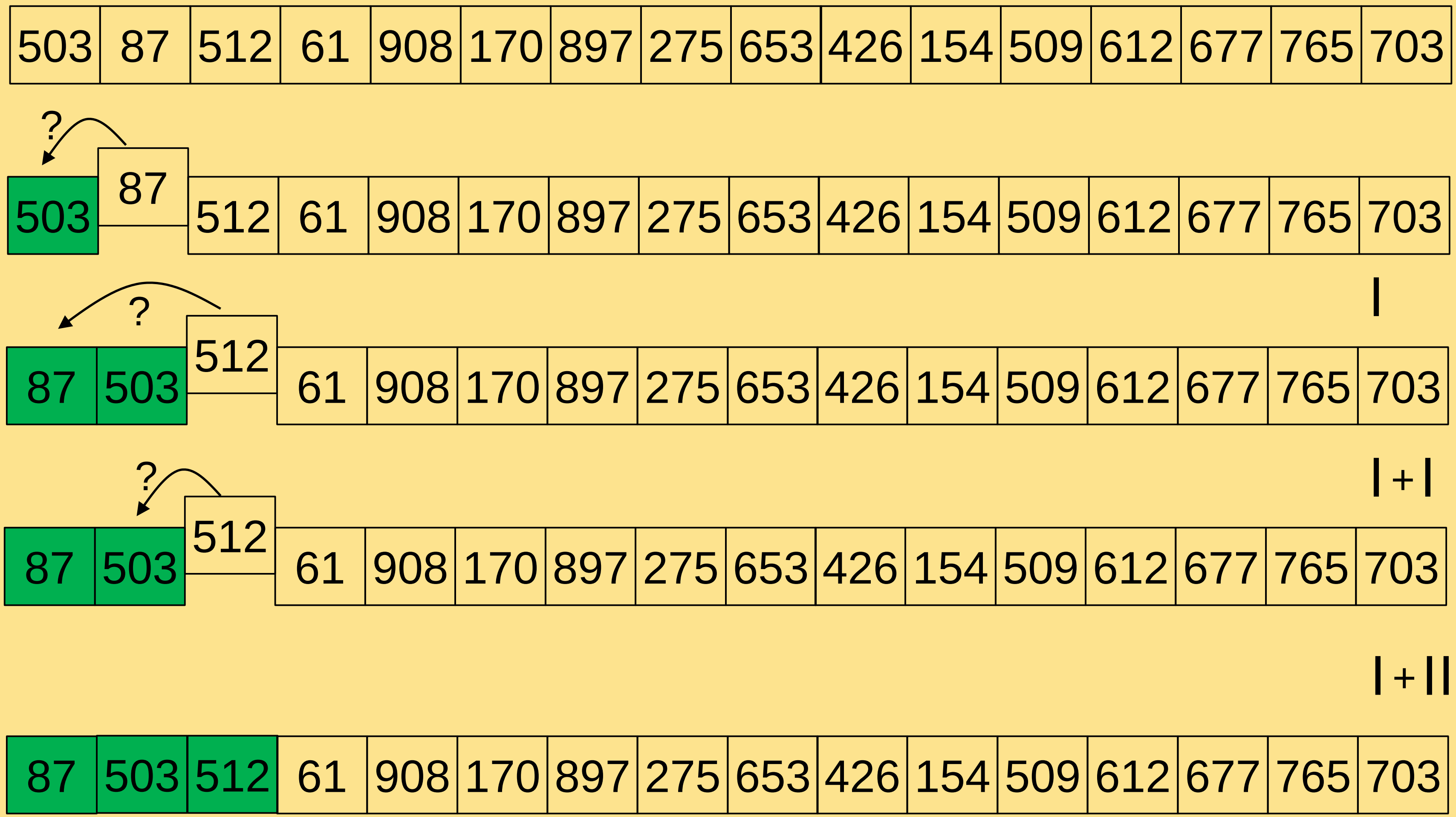


Announcement 2

- Starting from now, I will decrease the number of points for the home exercises by 1 for each hour, the solution is handed in too late
- Deadline: 11:59:59pm at the given date, Paris time
- Otherwise, it is unfair for the students who hand in on time

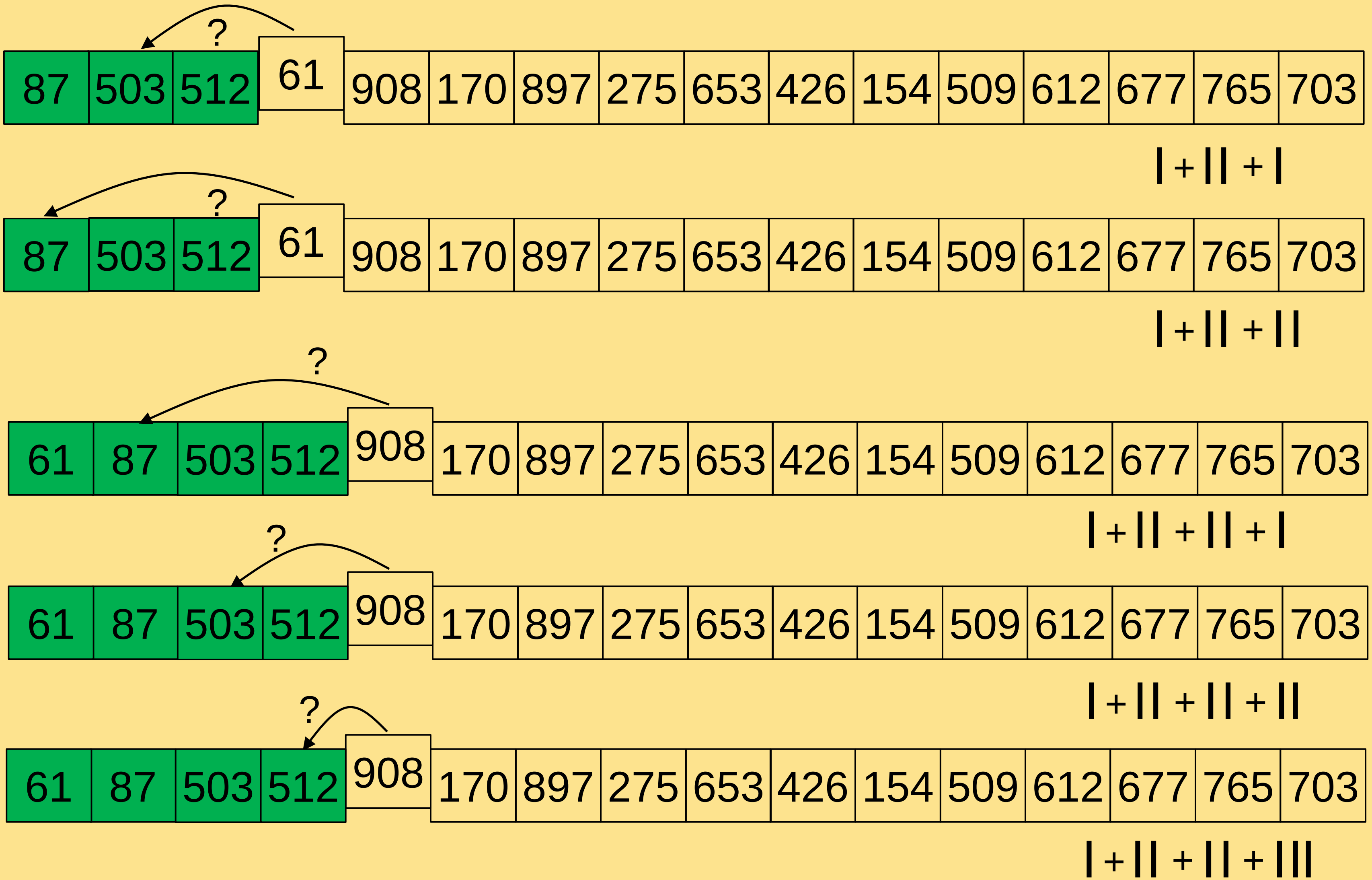
Discussion of Home Exercises

Exercise 1: Insertion Sort with binary search

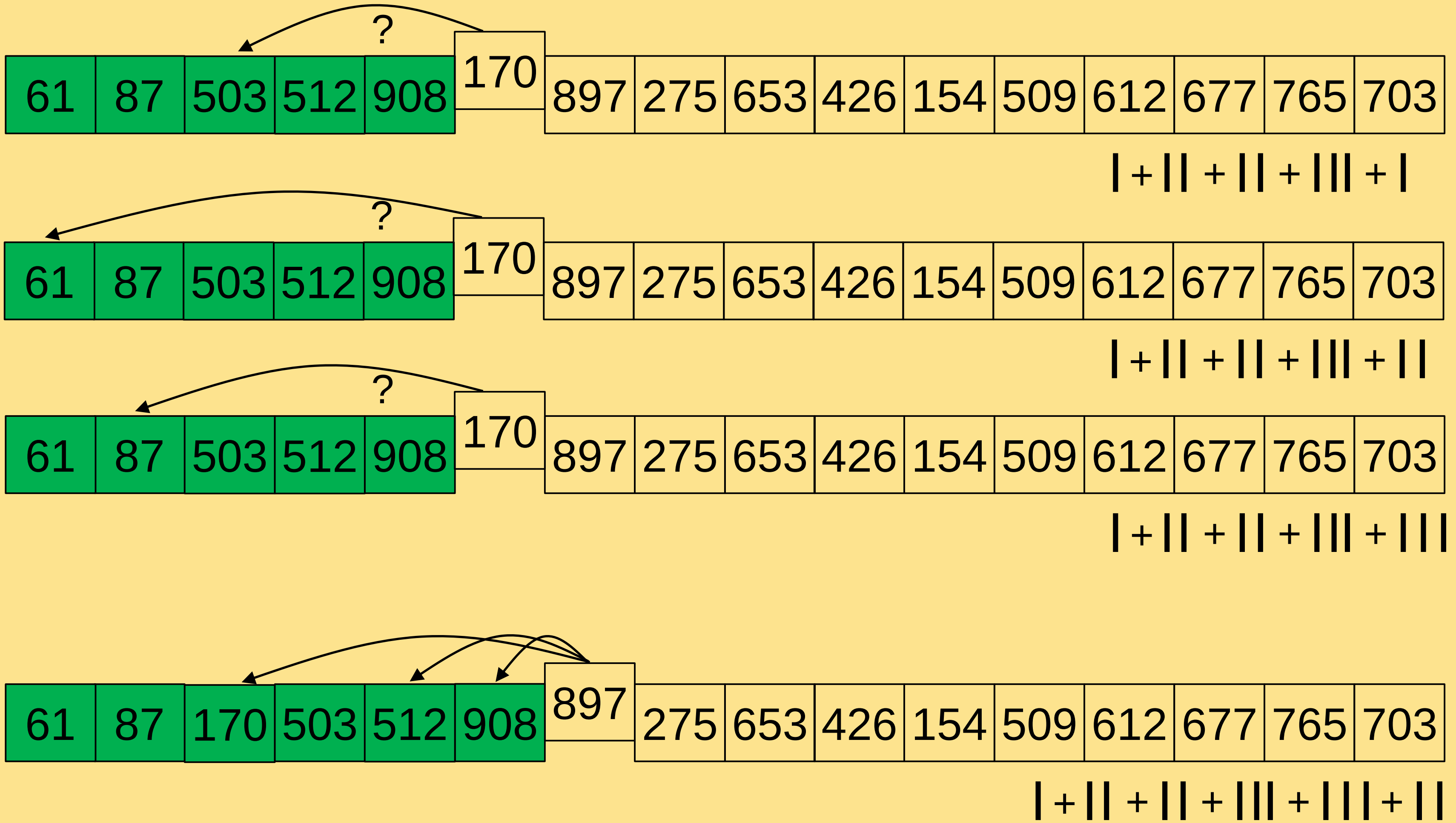


Discussion Home Exercise

Exercise 1: Insertion Sort with binary search

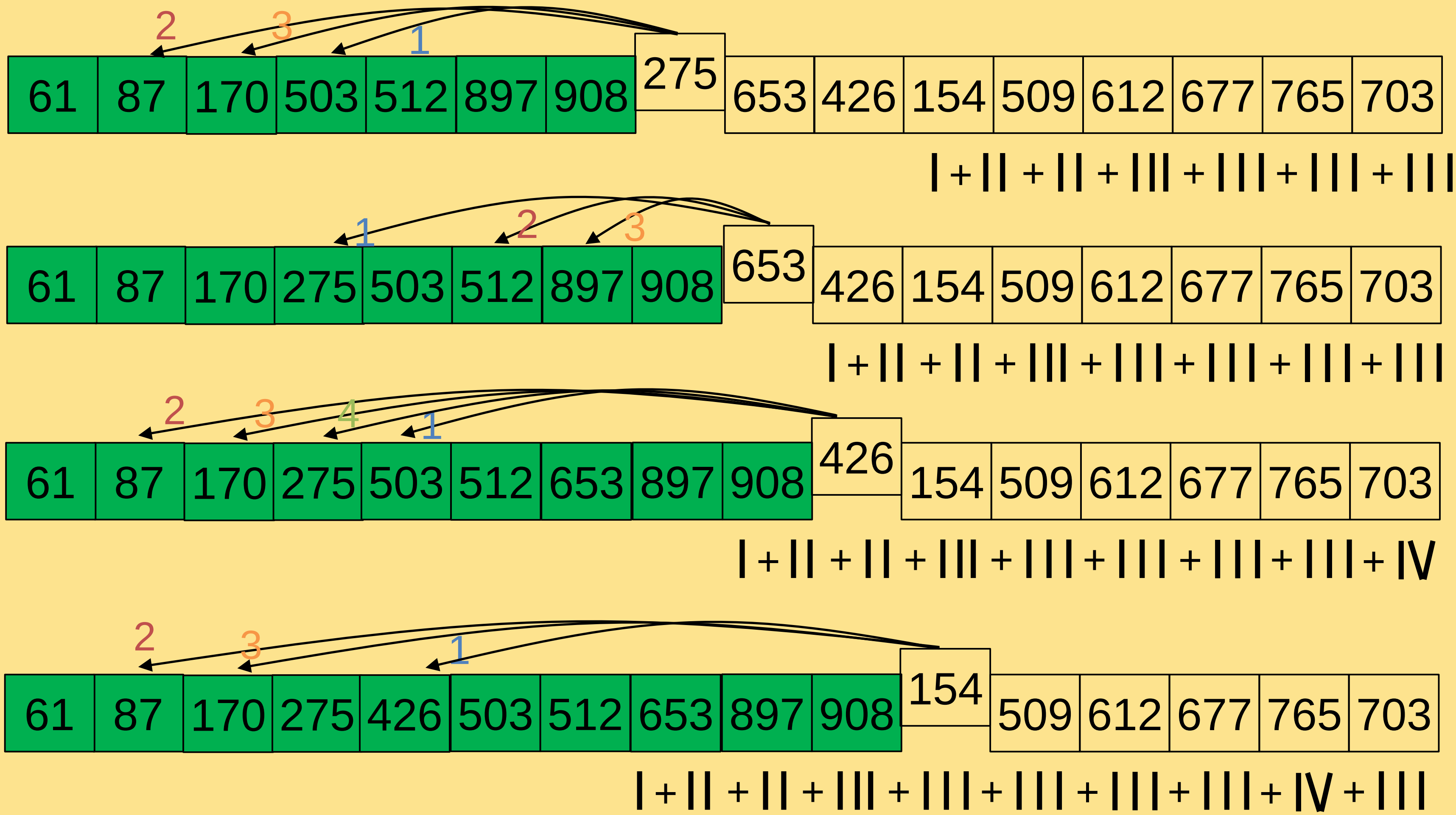


Exercise 1: Insertion Sort with binary search

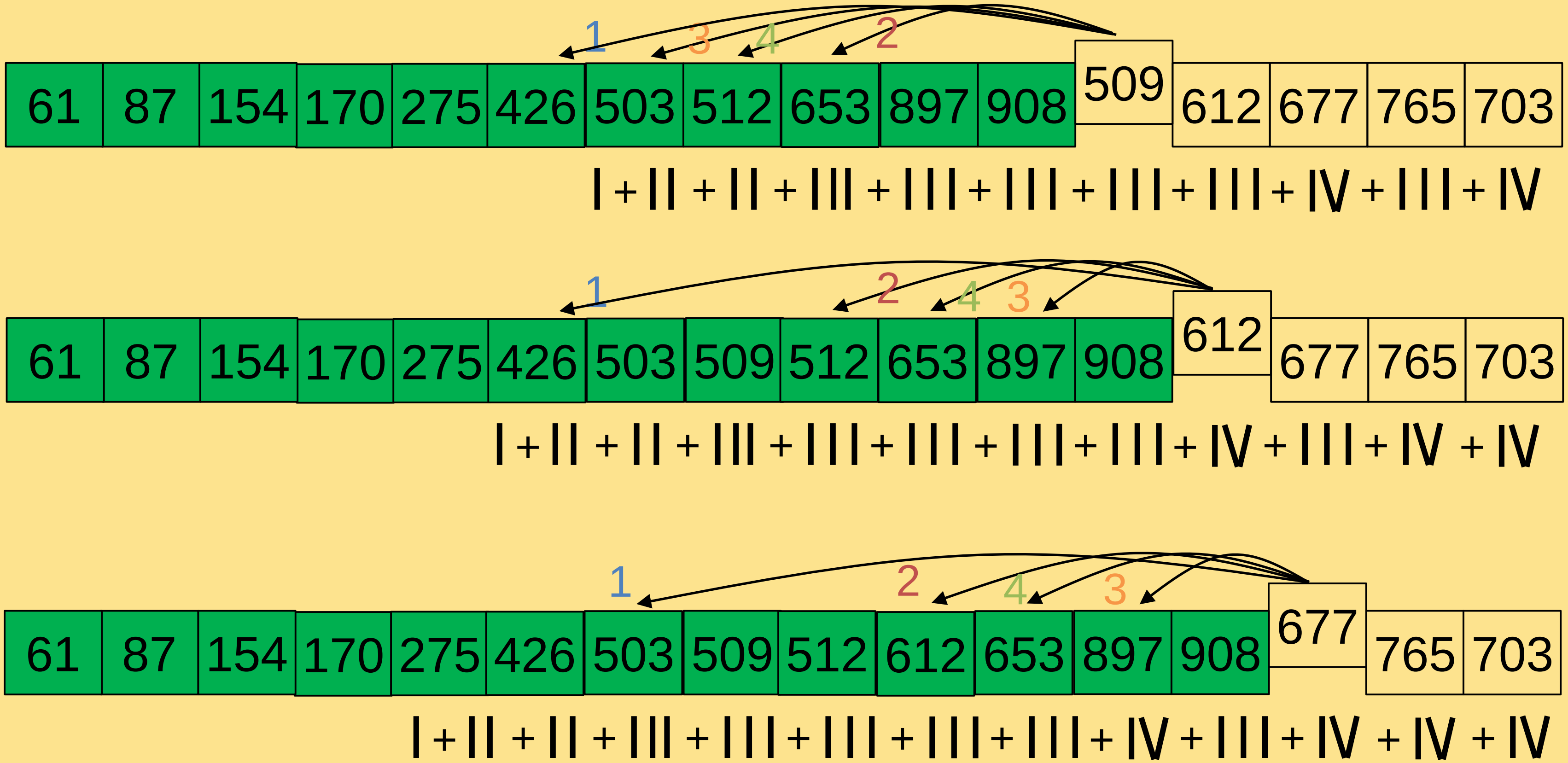


Discussion Home Exercise

Exercise 1: Insertion Sort with binary search



Exercise 1: Insertion Sort with binary search



Exercise 1: Insertion Sort with binary search

61	87	154	170	275	426	503	509	512	612	653	677	897	908	765

703

I + II + II + III + III + III + III + III + IV + III + IV + IV + IV + IV

61	87	154	170	275	426	503	509	512	612	653	677	765	897	908

703

I + II + II + III + III + III + III + III + IV + III + IV + IV + IV + IV + IV

61	87	154	170	275	426	503	509	512	612	653	677	703	765	897	908

In total: 47 comparisons

Exercise 2: Mergesort

503	87	512	61	908	170	897	275	653	426	154	509	612	677	765	703

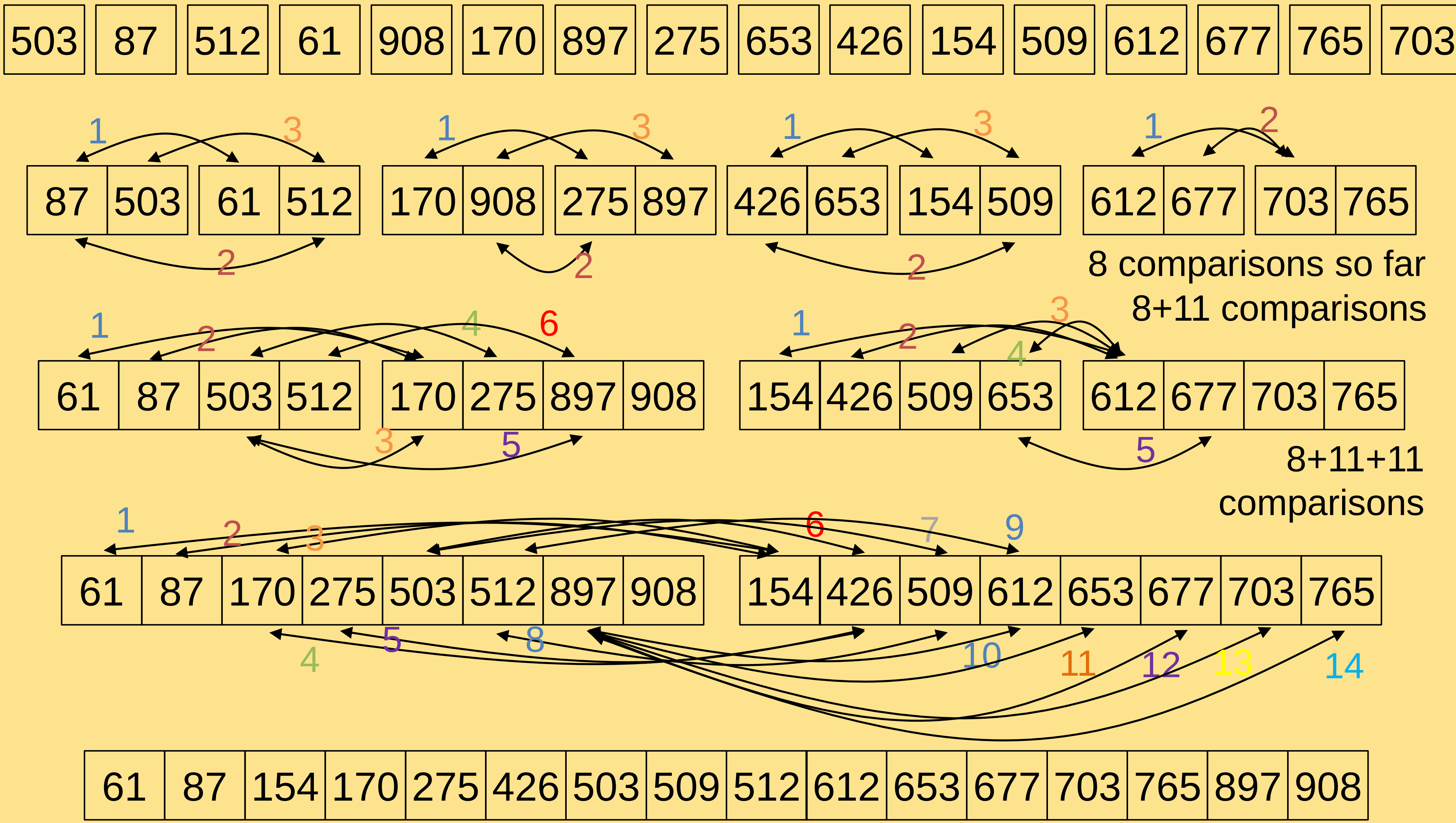
503	87	512	61	908	170	897	275		653	426	154	509	612	677	765	703

503	87	512	61		908	170	897	275		653	426	154	509		612	677	765	703

503	87		512	61		908	170		897	275		653	426		154	509		612	677		765	703

503	87	512	61	908	170	897	275	653	426	154	509	612	677	765	703

Exercise 2: Mergesort



8+11+11+14 = 44 comparisons in total

Discussion Home Exercise

Exercise 3: Finding the k largest elements with Merge-sort

a) Algorithmic changes

- put the larger values to the left [or start merging from right]
- can stop each merging after k elements

b) A simple upper bound on the runtime (when k is small)

- merge needs always at most $O(k)$ comparisons
- overall $\frac{\mathrm{n}}{2}+\frac{\mathrm{n}}{4}+\frac{\mathrm{n}}{8}+\cdots 2+1= \ O(n)$ merge steps needed
- in total: $O(k \cdot n)$ comparisons needed to find largest k elements

Discussion Home Exercise

Exercise 3: Finding the k largest elements with Merge-sort

c) A general upper bound on the runtime: $O(n \log k)$

- We assume for simplicity here that n and k are powers of 2
- In all splitting steps and in the first $\log k$ merging steps, there is no difference between the "new" and the original Mergesort algorithm (because the merging does not produce larger arrays than of length k)
- These merging steps take maximally O(n) comparisons each, which means O(n log k) in total.
  - Actually, in the ith merging step, there are $n/2^i$ merges of arrays of length 2^{i-1} which need maximally $2 \cdot 2^{i-1} - 1 = 2^i - 1$ comparisons each (hence $\frac{n}{2^i} \cdot (2^i - 1) = O(n)$ per merging step)
- More complicated is the analysis for the remaining log n – log k = log (n/k) merging steps…

Discussion Home Exercise

Exercise 3: Finding the k largest elements with Merge-sort

c) A general upper bound on the runtime: $O(n \log k)$

- […]
- Slightly more complicated is the analysis for the remaining $\log n - \log k = \log(n/k)$ merging steps:
  - In the ith-to-last merging step, we have 2^{i-1} merges of arrays of length k which need k comparisons each
  - summed over all $\log(n/k)$ remaining merge steps, we have

$$\sum_{i=1}^{\log(\frac{n}{k})} k \cdot 2^{i-1} = k \cdot \sum_{i=0}^{\log\left(\frac{n}{k}\right)-1} 2^i = k \cdot \left(\frac{1 - 2^{\log\left(\frac{n}{k}\right)}}{-1}\right) = n - k$$

comparisons because $\sum_{i=0}^{n} q^i = \frac{1-q^{n+1}}{1-q}$

- Thanks to Valentina, Rodolphe, and Arthur for the proof idea!

Recursive Algorithms (recap)

Recursive Algorithms

recursive algorithm/data structure/…
 = algorithm/data structure/… that calls/contains a self-reference

Examples:

- Mergesort
- Binary Search
- computing $n!$ (= $n \cdot (n-1)!$)
- there are also recursive data structures:
  - a linked list is defined as an element with data and pointer to another linked list
  - a tree: the root has other trees as children
- fractals are also recursive

back to last python exercise

Greedy Algorithms

Greedy Algorithms

From Wikipedia:

> "A *greedy algorithm* is an algorithm that follows the problem solving *heuristic* of making the locally optimal choice at each stage with the hope of finding a global optimum."

- Note: typically greedy algorithms do not find the global optimum
- We will see later when this is the case

Greedy Algorithms: Lecture Overview

- Example 1: Money Change
- Example 2: Packing Circles in Triangles
- Example 3: Minimal Spanning Trees (MST) and the algorithm of Kruskal
- Example 4: Bin Packing

Example 1: Money Change

Change-making problem

- Given n coins of distinct values $w_1=1, w_2, ..., w_n$ and a total change W (where $w_1, ..., w_n$, and W are integers).
- Minimize the total amount of coins Σx_i such that $\Sigma w_i x_i = W$ and where x_i is the number of times, coin i is given back as change.

Greedy Algorithm

Unless total change not reached:

add the largest coin which is not larger than the remaining amount to the change

Note: only optimal for standard coin sets, not for arbitrary ones!

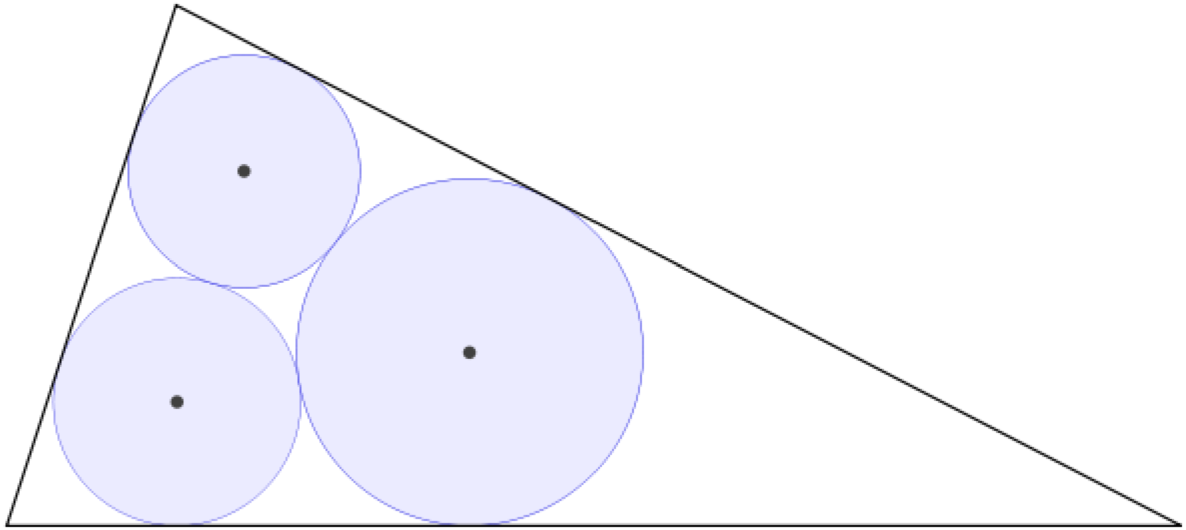
Related Problem:

finishing darts (from 501 to 0 with 9 darts)

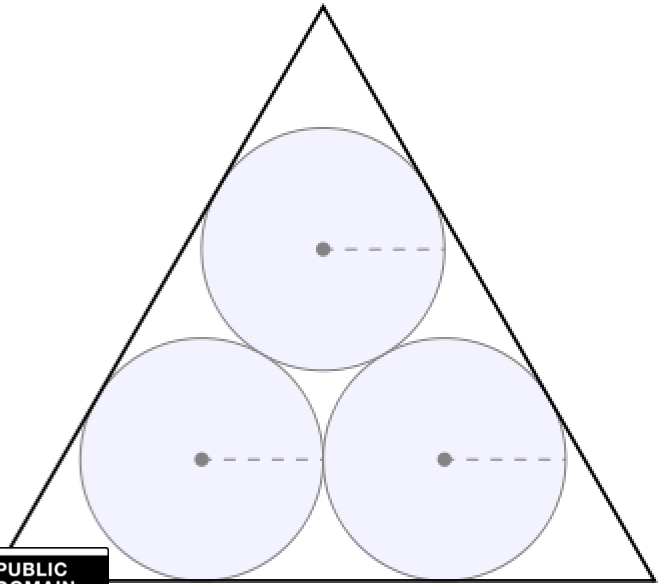
Example 2: Packing Circles in Triangles

G. F. Malfatti posed the following problem in 1803:

- how to cut three cylindrical columns out of a triangular prism of marble such that their total volume is maximized?
- his best solutions were so-called Malfatti circles in the triangular cross-section:
  - all circles are tangent to each other
  - two of them are tangent to each side of the triangle

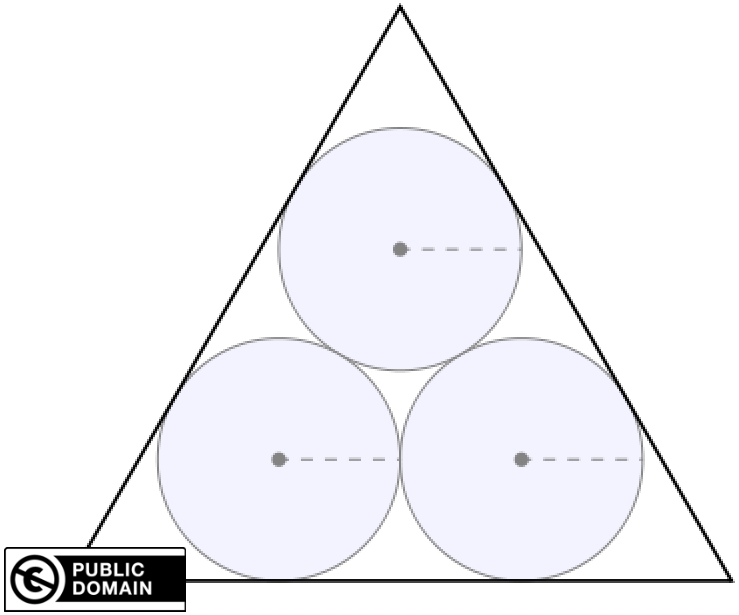


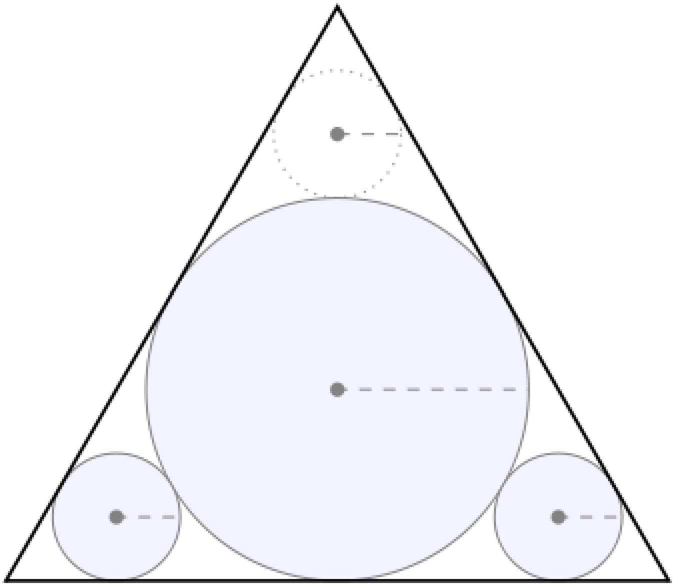
Example 2: Packing Circles in Triangles



What would a greedy algorithm do?

Example 2: Packing Circles in Triangles





What would a greedy algorithm do?

Note that Zalgaller and Los' showed in 1994 that the greedy algorithm is optimal [1]

[1] Zalgaller, V.A.; Los', G.A. (1994), "The solution of Malfatti's problem", *Journal of Mathematical Sciences* **72** (4): 3163–3177, doi:10.1007/BF01249514.

Example 3: Minimal Spanning Trees (MST)

Outline:

- reminder of problem definition
- Kruskal's algorithm
  - including correctness proofs and analysis of running time

MST: Reminder of Problem Definition

A *spanning tree* of a connected graph G is a tree in G which contains all vertices of G

Minimum Spanning Tree Problem (MST):

Given a (connected) graph G=(V,E) with edge weights w_i for each edge e_i. Find a spanning tree T that minimizes the weights of the contained edges, i.e. where

$$\sum_{e_i \in T} w_i$$

is minimized.

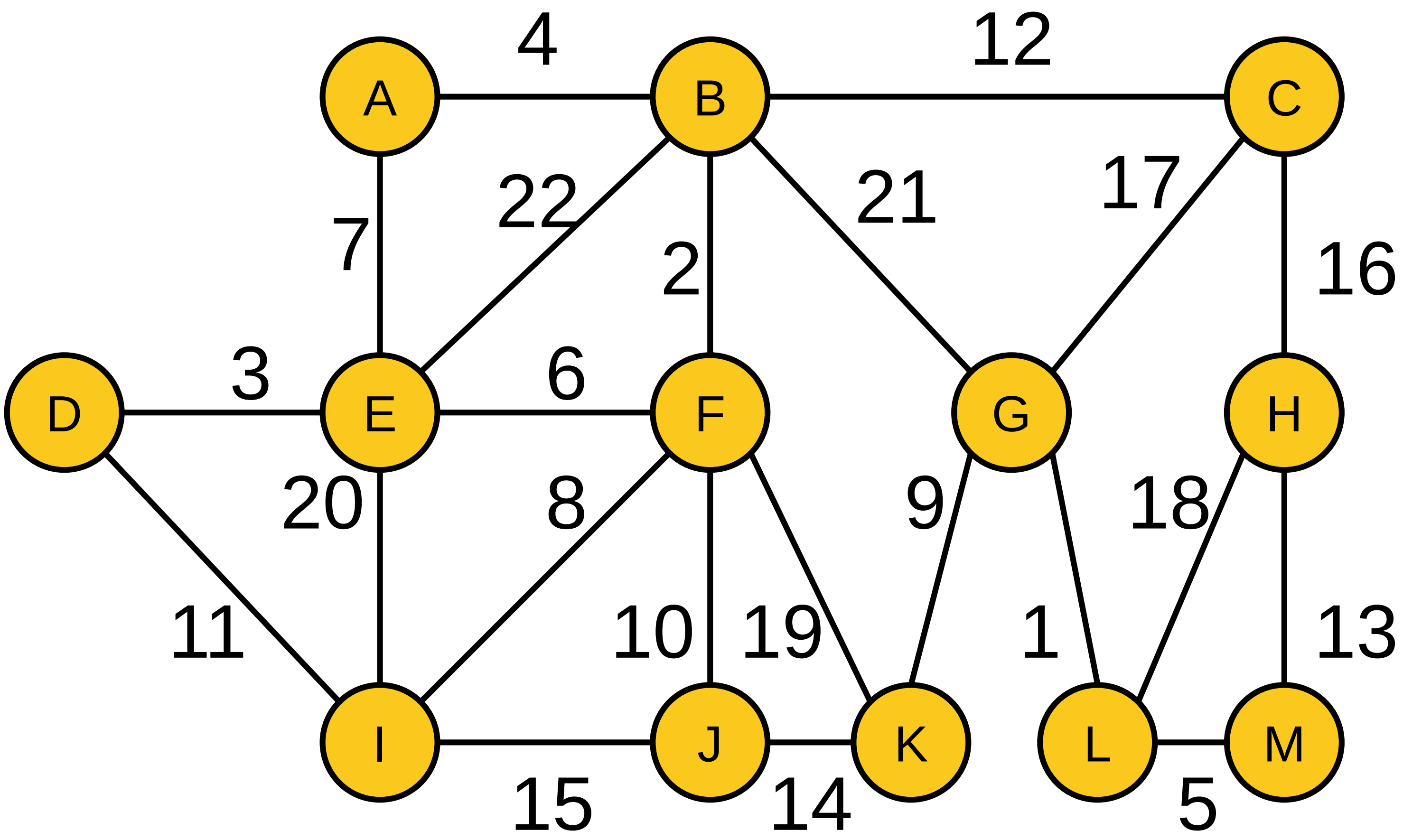
Kruskal's Algorithm

Algorithm, see [1]

- Create forest F = (V,{}) with n components and no edge
- Put sorted edges (such that w.l.o.g. $w_1 < w_2 < \ldots < w_{|E|}$) into set S
- While S non-empty and F not spanning:
  - delete cheapest edge from S
  - add it to F if no cycle is introduced

[1] Kruskal, J. B. (1956). "On the shortest spanning subtree of a graph and the traveling salesman problem". *Proceedings of the American Mathematical Society* **7**: 48–50. doi:10.1090/S0002-9939-1956-0078686-7

Kruskal's Algorithm: Example



Kruskal's Algorithm: Runtime Considerations

First question: how to implement the algorithm?

- sorting of edges needs $O(|E| \log |E|)$

Algorithm

Create forest F = (V,{}) with n components and no edge
Put sorted edges (such that w.l.o.g. $w_1 < w_2 < ... < w_{|E|}$) into set S
While S non-empty and F not spanning:
  delete cheapest edge from S
  add it to F if no cycle is introduced

simple

?

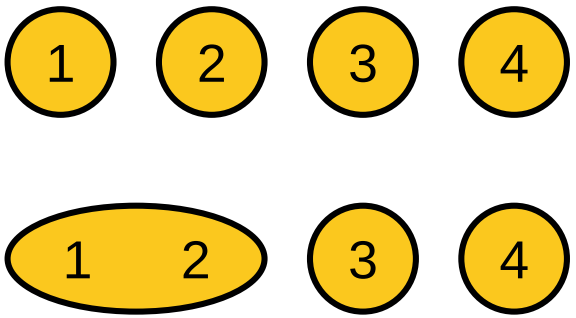
forest implementation:
Disjoint-set data structure

Disjoint-set Data Structure (“Union&Find”)

Data structure: ground set 1...N grouped to disjoint sets

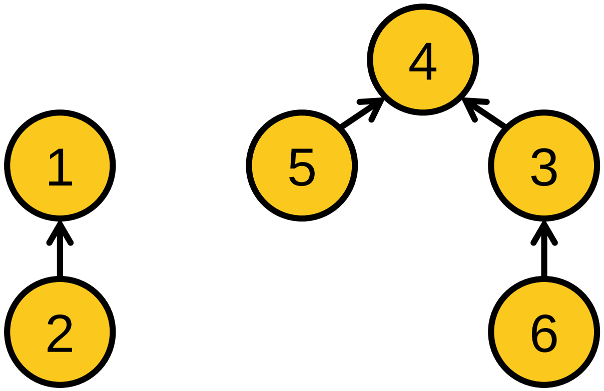
Operations:

- FIND(i): to which set (“tree”) does i belong?
- UNION(i,j): union the sets of i and j!
  (“join the two trees of i and j”)



Implemented as trees:

- UNION(T1, T2): hang root node of smaller tree under root node of larger tree (constant time), thus
- FIND(u): traverse tree from u to root (to return a representative of u’s set) takes logarithmic time in total number of nodes



Implementation of Kruskal's Algorithm

Algorithm, rewritten with UNION-FIND:

- Create initial disjoint-set data structure, i.e. for each vertex v_i, store v_i as representative of its set
- Create empty forest F = {}
- Sort edges such that w.l.o.g. $w_1 < w_2 < ... < w_{|E|}$
- for each edge e_i={u,v} starting from i=1:
  - if FIND(u) ≠ FIND(v): # no cycle introduced
    - F = F ∪ {{u,v}}
    - UNION(u,v)
- return F

Back to Runtime Considerations

- Sorting of edges needs O(|E| log |E|)
- forest: **Disjoint-set data structure**
  - initialization: O(|V|)
  - log |V| to find out whether the minimum-cost edge {u,v} connects two sets (no cycle induced) or is within a set (cycle would be induced)
  - 2x FIND + potential UNION needs to be done O(|E|) times
  - total O(|E| log |V|)
- Overall: O(|E| log |E|)

slides with a blueish background have not been discussed in class and, thus, are not part of the final exam

Kruskal's Algorithm: Proof of Correctness

Two parts needed:

❶ Algo always produces a spanning tree

final F contains no cycle and is connected by definition ✓

❷ Algo always produces a *minimum* spanning tree

- argument by induction
- P: If *F* is forest at a given stage of the algorithm, then there is some minimum spanning tree that contains *F*.
- clearly true for F = (V, {})
- assume that P holds when new edge e is added to F and be T a MST that contains F
  - if e in T, fine
  - if e not in T: T + e has cycle C with edge f in C but not in F (otherwise e would have introduced a cycle in F)
    - now T – f + e is a tree with same weight as T (since T is a MST and f was not chosen to F)
    - hence T – f + e is MST including T + e (i.e. P holds) ✓

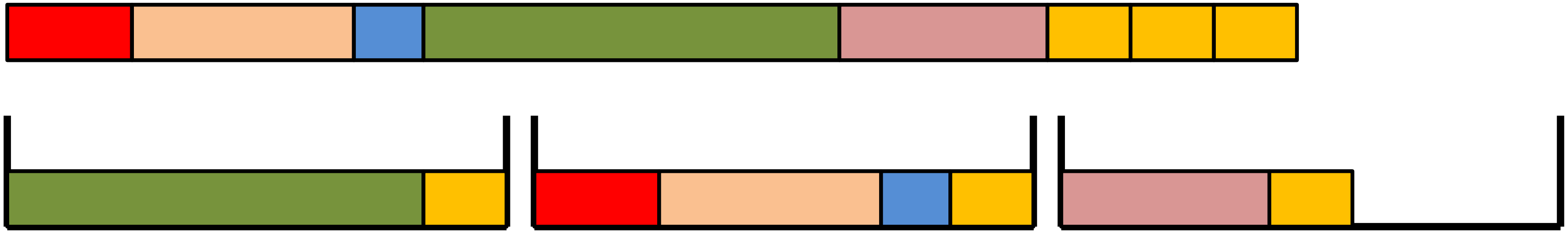
Another Greedy Algorithm for MST

- Another greedy approach to the MST problem is Prim's algorithm
- Somehow like the one of Kruskal but:
  - always keeps a tree instead of a forest
  - thus, take always the cheapest edge which connects to the current tree
- Runtime more or less the same for both algorithms, but analysis of Prim's algorithm a bit more involved because it needs (even) more complicated data structures to achieve it (hence not shown here)

Example 3: Bin Packing (BP)

Bin Packing Problem

Given a set of n items with sizes a_1, a_2, ..., a_n. Find an assignment of the a_i's to bins of size V such that the number of bins is minimal and the sum of the sizes of all items assigned to each bin is $\leq$ V.



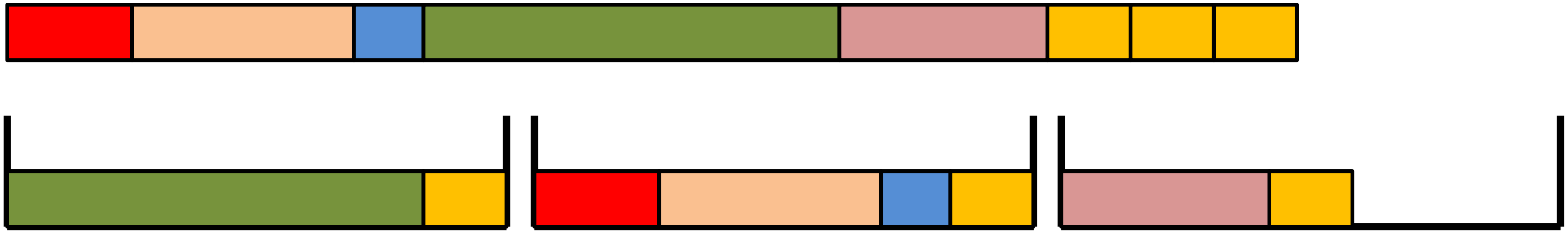
Applications

similar to multiprocessor scheduling of n jobs to m processors

Example 3: Bin Packing (BP)

Bin Packing Problem

Given a set of n items with sizes $a_1, a_2, ..., a_n$. Find an assignment of the a_i's to bins of size V such that the number of bins is minimal and the sum of the sizes of all items assigned to each bin is $\leq V$.



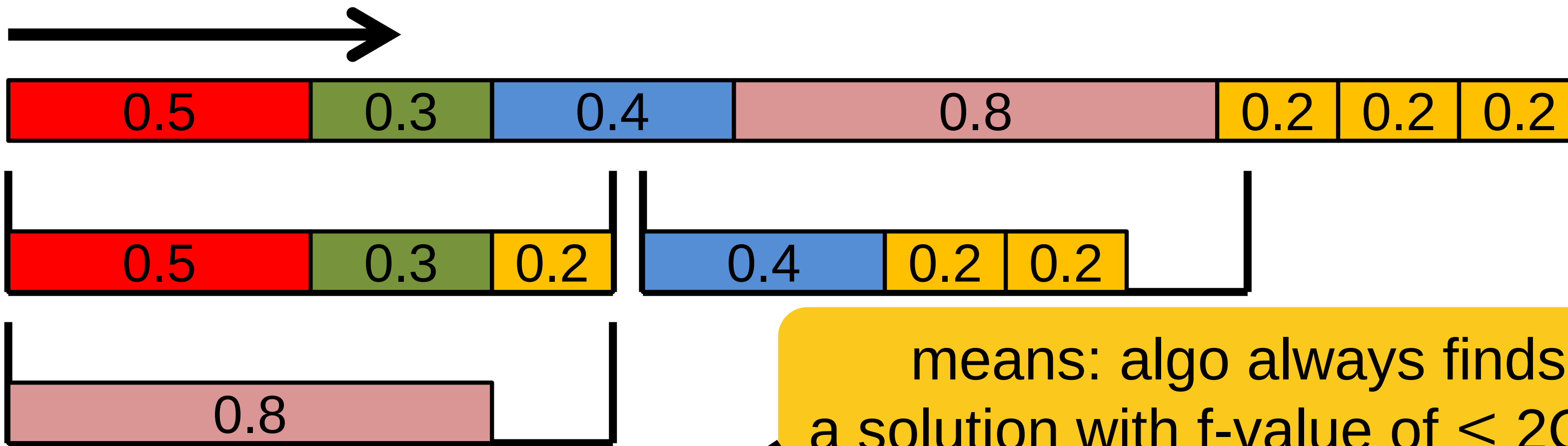
Known Facts

- no optimization algorithm reaches a better than 3/2 approximation in polynomial time (not shown here)
- greedy first-fit approach already yields an approximation algorithm with approximation ratio of 2

First-Fit Approach

First-Fit Algorithm

- without sorting the items do:
  - put each item into the first bin where it fits
  - if it does not fit anywhere, open a new bin



means: algo always finds a solution with f-value of ≤ 2OPT

Theorem: First-Fit algorithm is a 2-approximation algorithm

Proof: Assume First Fit uses m bins. Then, at least m-1 bins are more than half full (otherwise, move items).

$$\text{OPT} > \frac{m-1}{2} \iff 2\text{OPT} > m-1 \implies 2\text{OPT} \geq m$$

↑because m and OPT are integer

Conclusion Greedy Algorithms I

What we have seen so far:

- three problems where a greedy algorithm was optimal
  - money change
  - circle packing
  - minimum spanning tree (Kruskal’s algorithm)
- but also: greedy not always optimal
  - see the example of bin packing
  - this is true in particular for so-called NP-hard problems

Obvious Question: when is greedy good?

Answer: if the problem is a matroid (not covered here)

From Wikipedia: [...] a matroid is a structure that captures and generalizes the notion of linear independence in vector spaces. There are many equivalent ways to define a matroid, the most significant being in terms of independent sets, bases, circuits, closed sets or flats, closure operators, and rank functions.

Conclusions Greedy Algorithms II

I hope it became clear...

...what a greedy algorithm is

...that it not always results in the optimal solution

...but that it does if and only if the problem is a matroid